

**LINUX MAGAZINE** • [subscribe](#) • [advertise](#) • [customer service](#) • [back issues](#) • [feedback](#) • [contacts](#)

## Newsletter




### J/XFS Devices Services

devices services for ongoing J/XFS kernel

### Solaris Kernel Tuning

Automated recommendation explanations for Solaris, AIX

Ad

Linux Magazine / July 1999 / FEATURES  
*Secrets Inside the Linux Kernel Revealed*

[<< prev](#) [page 1](#) [2](#) [3](#) [4](#) [next >>](#)

Ads by Goooooogle

#### Dsp Kernel

Operating Systems  
Intel® Developer  
Services  
[www.intel.com](http://www.intel.com)

#### Understand Linux Kernel

Buy Daniel P.  
Bovet's book Now  
at 30% off.  
associate  
[www.amazon.com](http://www.amazon.com)

#### Linux

Rocks and Rolls!  
[linspire.com](http://linspire.com)

#### 4-way Itanium® 2 Systems

4U: 1.5GHz w.  
6MB Cache, 32GB  
RAM Configure &  
Price online at ION  
[itanium.ioncomputer.com](http://itanium.ioncomputer.com)

## Virtual Memory

Earlier, I said that the virtual memory subsystem in the kernel w memory protection (preventing processes from interfering with c memory) and paging (allowing memory pages to be temporarily disk). This is easily one of the most complex aspects of the Linu and it bears taking a closer look at. Understanding how virtual m Linux is the key to understanding many other features, such as tl

This is a topic which has consumed a fair number of pages in bo advanced operating systems textbooks (not to mention a slew of and articles in magazines such as *Glamour*), so I can't expect to bulk of virtual memory management in this article. Hopefully, tl you a clear picture of what the various pieces are so you can loo more detail elsewhere. Different operating systems implement v management in very different ways, so my treatment here will b Linux. It's also helpful to focus on a particular hardware architec Linux relies heavily (as do other operating systems) on the mem support provided by the hardware. In this case, I'll focus on the I discussion is similar for other systems.

The very meaning of the term virtual memory is that an applicat the illusion that it has access to a much larger amount of memor present on the system. Ideally, we'd like the application to believ entire range of memory addresses allowed by the CPU -- which  $2^{32}$  bytes, or 4 gigabytes of virtual memory. (This is because eve "virtual address", is stored in 32 bits and each bit only has 2 pos:  $2^{32}$ , then, is the range of virtual memory which can be addresser pointer. We can write this range, in hexadecimal, as 0x 00000000 Very few systems, however, have 4 gigabytes of physical memo addition, multiple applications may be running on the system at each application to be able to access the entire 4-gigabyte "virtu: without interfering with one another. How are we going to accor

## Sections

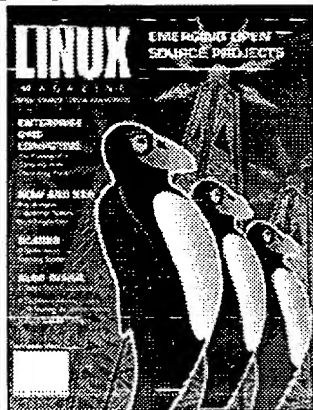
[Editorial](#)

[Newbies](#)

Luckily, the Intel x86 architecture (as well as most other modern architectures) includes hardware support for implementing virtu:

[Reviews](#)[Shutdown](#)[How To](#)[LAMP Post](#)[Power Tools](#)[Guru Guidance](#)[On The Desktop](#)[Best Defense](#)[Developer's Den](#)[Java Matters](#)[Extreme Linux](#)[Compile Time](#)[Perl of Wisdom](#)[Gearheads Only](#)[Project of the Month](#)[Who's Who](#)[Downloads](#)

## On Stands Now



## Indexes

[Issue Archive](#)[Author Index](#)

RSS feed:

## Subscribe Now!

Name: Address: Address2: City: State:  XX (US only)Zip: Email: ☒ 12 issues for \$29.95☐ 24 issues for \$54.95

form of a "memory management unit", or MMU. The MMU is responsible for translating virtual memory accesses made by user programs into accesses of the actual RAM in the machine. Of course, the MMU needs the operating system to do this -- which is where the Linux kernel subsystem comes into play. Before we get into all of that, however, let's see what happens when a user program reads or writes an address in

## Translating virtual addresses

Let's say that a user process (such as Emacs) wants to read or write a virtual address 0x48b32f00 (this address is meaningless to a human, of course, but to Emacs it might represent the current position of the window). In order to translate this memory reference into an actual physical memory address, the MMU uses a special set of structures, called the page tables, to specify for each page (4 kilobytes) of virtual memory what physical address (if any) should be used. We can think of a particular page containing the following information:

```
virtual page address -> physical page address (+ other information)
```

So, there might be an entry in the page tables which looks like:

```
0x48b32000 -> 0x0028a000 (+ other information)
```

Note that the MMU deals with memory in page-sized chunks, so the last 3 digits (in hexadecimal) of the virtual and physical page addresses are always '000'. The last three digits of an address (such as 0x48b32f00) represent the offset into the page which the user is accessing; in this case that would be 0xf00. The MMU adds the page offset to the physical page address found in the page tables to produce a complete physical memory address -- *voila!*

Certainly not every virtual address in the entire 4 GB range corresponds to a physical memory address -- unless one has 4 GB of memory installed. These page table entries might contain a physical page address which is "invalid", meaning that there is no physical page corresponding to that virtual address. When an invalid entry is read by the MMU, a page fault occurs. We'll talk about this case later.

## Page tables and the TLB

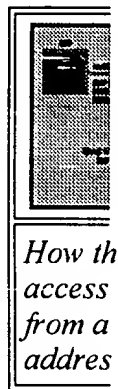
Note that the page tables are actually stored in memory themselves. This is an interesting issue, as it means that the MMU might have to consult the page tables in order to look up something else in the page tables. This is the case because of the fact that there is not just a single page table -- rather, there are thousands of page tables which must be traversed by the MMU in order to translate a virtual address into a physical address. This is done for space reasons; a single page table entry actually consumes four bytes. If we have

Non-US orders? Click here for delivery to:  
[Canada](#) • [Mexico](#) • [Other](#)

entry per page of virtual address space, that's  $((4 \text{ Gb}/4 \text{ Kb}) * 4 \text{ b})$  megabytes of memory, just to hold the page tables for a single u multiple levels of page tables actually allows the hardware to sw page tables out to disk when they're not being used -- we warnec going to get hairy! For now, though, don't worry about it -- it su that there is a single (in-memory) page table being consulted by every memory reference.

If the MMU is performing multiple memory operations just to tr address for another memory operation, clearly this is going to be performance. To remedy this problem, the MMU hardware inclu for virtual-to-physical memory translations, called the translation or TLB. The TLB can be thought of as containing a very small p tables in very fast RAM to speed up MMU address translations.

Figure 2 shows what happens during a single memory access by a user application. On the upper left of the figure we have the CPU, from which a user process wishes to read the virtual address 0x48b32f00. First, the TLB is consulted, and if a translation is found there, the physical address is used to access memory directly. (We've drawn the TLB as being separate from the CPU itself, but it's actually part of the processor in most cases.) If the TLB does not contain a mapping for the given virtual address, the MMU must then perform the arduous task of looking up the address in the page tables. However, the result of this lookup will be saved in the TL increasing performance if another address on the same page is a



So far I've talked about the Intel x86's MMU hardware, not about Clearly, the kernel isn't involved every time a virtual address is t would be far too slow. So, what does the kernel have to do with management?

The first job of the kernel is to set up and maintain the page tabl will traverse when translating addresses. This requires the kerne how physical memory is laid out, to allocate portions of it to var processes, and to create page table entries allowing virtual addre process to eventually map onto physical RAM.

[<< prev](#) [page 1](#) [2](#) [3](#) [4](#) [next >>](#)

[Linux Magazine / July 1999 / FEATURES](#)  
*Secrets Inside the Linux Kernel Revealed*

[LINUX MAGAZINE](#) • [subscribe](#) • [advertise](#) • [customer service](#) • [back issues](#) • [feedback](#) • [contacts](#)

**This Page Blank (uspto)**